Interested in learning more about security?

SANS Institute InfoSec Reading Room

PAM - It's NOT The Non-Stick Cooking Spray

To combat brute force password cracking attempts in Unix, shadow passwords were introduced. While the user specific information remained in the /etc/passwd file, the encrypted password and other sensitive information was moved to an /etc/shadow file that was only readable by "root". This change in the file structure meant the programs or services used for user authentication had to change as well. A recompile of the various application and service source codes to institute the changes was required. In the example above...

PAM – It's NOT The Non-Stick Cooking Spray

For those of us who will admit being old enough to remember the good old days of Unix, you will recollect that most authentication involved a password scheme of some kind. Regardless of what system was in place, the necessary authentication information was derived from a single flat file. In most cases, the file was the */etc/passwd* file. The result was always the same; the user was denied or granted access. Programs such as rcp, su, ftp, or xlock, among others, needed to use the same file for their authentication. The problem with the password file in those days was that it had to be readable by everyone. This easy access to the information in the file meant both the good and bad guys could examine the passwords. To combat the brute force password cracking attempts, shadow passwords were introduced. So, while the user specific information remained in the */etc/passwd* file, the encrypted password and other sensitive information was moved to an */etc/shadow* file that was only readable by "root". This change in the file structure meant the programs or services used for user authentication had to change as well. A recompile of the various application and service source codes to institute the changes was required.

In the example above, changing your method of authentication meant that every single program that relied on accessing that information had to be modified and recompiled in order to use the new method. What happened if you missed one? With today's Unix variants, there is a new solution – pluggable authentication modules or PAM. PAM aware applications and services allow transparent authentication, logging, or the setting of limits for users, regardless of the method used. PAM eliminates the need to replace binaries or recompile code. Pam can be a common authentication scheme that is used with a variety of applications and services. This paper will introduce the reader to PAM. It will provide an overview of PAM, it's origins, what PAM is, and an example of how it works.

In The Beginning

PAM was originally developed by Sun Microsystems ™ and adopted by OSF for inclusion in CDE/Motif. The user community was introduced to PAM in Solaris 2.3 as an undocumented feature. Sun did little more with the development of PAM in subsequent releases. The versatility of PAM caught the attention of the open source community. It was the Linux community, in particular, that has popularized PAM.

What is PAM?

Quoted from the Linux-PAM Systems Administrators Guide: *"It is the purpose of Linux-PAM project to separate the development of privilege granting software from the development of secure and appropriate authentication schemes. This is accomplished by providing a library of functions that an application may use to request that a user be authenticated."*

Said another way; pluggable authentication modules provide the backbone of most authentication in modern Linux systems. PAM provides a middle layer to authentication, unlike the example stated in the introduction where changing your authentication scheme meant recompiling every binary that relied on that design. The applications and services are not aware of the actual authentication method being used and the result is a much more flexible system. As long as the application or service supports PAM, the system administrator can choose how any method of authentication.

PAM's Format

The PAM configuration files are special files that are consulted each time an application or service that is "PAM aware" is executed. The location or names of these files depend on the distribution of LINUX you are using. You will either find a directory /etc/pam.d which has separate configuration files for each application and service or there a single file /etc/pam.conf which has all configuration information for all applications and services. The examples shown in this paper will be based on the Redhat distribution and will use the /etc/pam.d directory structure. The files in the /etc/pam.d directory determine which PAM modules to invoke and what to do based on the results of each execution. The result of success, failure, or data is then passed to the next module.

Below is sample output from the *ls /etc/pam.d* command:

chfn	isdn-config	linuxconf-pair	poweroff	rsh	xdm
chsh	kbdrate	login	ppp	samba xlock	
ftp	kde	mcserv reboot	shutdown	xscreensaver	
gdm	kpackage	other	rexec	sshd	xserver
gnorpm-auth	kppp	passwd	rlogin	su	
halt	linuxconf	pop	rp3-config	vlock	

Each of these files contains one or more lines or records which define the authentication scheme for that particular application or service. Please notice in the sample directory listing the file named "*other*". "Other" is a special default file that applies to any service or application that is not specifically named in this directory. By default, this file calls the module pam_deny.so which issues a default deny or failure. This module does no logging that would alert the system administrator of its execution. It is highly desirable that you modify your "other" file in the /etc/pam.d directory to include the pam_warn.so module. By adding this module, a suitable warning would be issued and proper action can then be taken to create an application or service specific file.

The structure of the files found in the /etc/pam.d follows. Please note that each line is considered a record and the fields can be separated by either spaces or tabs.

module-typecontrol-flag module-path[argument(s…)]

module-type Specifies the functional area of the record. The module-type can be *"auth", "account", "password", or "session"*. These types are defined as:

auth	Instructs the application to prompt the user for identification and authentication information, such as the username and password
account	Checks aspects of the user's account, such as password aging, time limits, and system resource limits
session	Sets up the environment and establishes logging
password	Maintains and updates user authentication information, such as passwords

control-flag Determines the behavior of the authentication process based on the results of the module. The control-flag also determines the relative importance of modules in the list. This field can also accept instructions based on the return code of the module. The standard control-flags are:

required	The module must succeed in order for the service to be granted. If the record containing this flag is one of a "stack" of records with the same type field, then all the other modules will be executed, even if this module fails.
requisite	The module must succeed. If the module fails, no other modules are executed and PAM returns a failure or denies immediately.
sufficient	Checks done by a module are ignored if they fail. But, if a sufficient flagged module is successfully checked and no required flagged modules above it have failed, then no other modules of this module-type are checked and this module-type is considered to have successfully been checked.
optional	The module in the record does not need to succeed. The module's success or failure is only important if it is the only record for that module-type.

module-path Specifies the PAM module to be executed. Each module performs a specific task and returns information that is used to determine the success or failure of the authentication or the data is used by one of the other called modules. Depending on your flavor of Unix the modules that PAM uses can be found in either */usr/lib/security* or */lib/security*.

arguments Optional arguments to be passed to the module. Some arguments are module specific . Examples of some generic arguments are:

debug	Uses the syslog facility to log debugging information to the systems log files
no_warn	Tells the module not to pass warning messages to the application
use_first_pass	This tells the module not to ask for the user password. The password should be obtained from the previous auth module. If that module did not succeed, then the user is prompted for a password.

It should be noted for completeness that if your distribution of LINUX uses /etc/pam.conf, the configuration lines in this file are slightly different. The service or application name will be the first field followed by the module-type, control-flag and so on. For the complete PAM module reference, please consider the Linux-PAM System

Administrator's Guide. The on-line guide contains up-to-date information and serves as a good reference.

How PAM Works

The following will walk you through one example of how PAM works. Assume the user has initiated a rlogin process to a local Redhat Linux system. The rlogin process is PAM aware and so it checks the /etc/pam.d directory to determine how to proceed. For our example we will use the /etc/pam.d/rlogin file seen below.

```
#%PAM-1.0
auth        sufficient    /lib/security/pam_rhosts_auth.so
auth        required      /lib/security/pam_securetty.so
auth        required      /lib/security/pam_pwdb.so shadow nullok
auth        required      /lib/security/pam_nologin.so
account     required      /lib/security/pam_pwdb.so
password    required      /lib/security/pam_cracklib.so
password    required      /lib/security/pam_pwdb.so shadow nullok use_authtok
session     required      /lib/security/pam_pwdb.so
```

This configuration file first runs the pam_rhosts_auth.so module. It is checking to see if the user has a ~/.rhosts file or the system has a /etc/hosts.equiv . If so, since its control-flag is set to "sufficient", it can grant the user access by itself. None of the subsequent auth module-types would have to be checked. It is usually recommended as part of good security practice to comment out or remove the pam_rhosts_auth.so module in all your services files.

For our example, there is no ~/.rhosts file or /etc/hosts.equiv present. The next module to run is pam_securetty.so is run. This is checking to see if the user is logging in as root. If they are logging in as root, it checks the /etc/securetty file to see if the terminal they are using is listed as being secure. If not, then pam_securetty will deny access.

The third module called in the auth section uses the regular password-checking module. The arguments *"shadow"* and *"nullok"* will cause the pam_pwdb module to check the /etc/shadow file for the hashed password. In addition, if the user had a null password set, it will still allow them access. Normally, pam_pwdb will return a failure if there was no password.

Finally, the last line in the authentication stack calls pam_nologin.so. If you have a /etc/nologin file, it will show the user the contents of this file and disallow access.

Once the process has passed "auth", the "account" module verifies that the user has a valid account. It checks the password age and whether or not the user needs to be warned about the need to change their password. If the user must select a new password, control is passed to the password module-type for this purpose. The module returns success if the user has a valid account.

The next two lines handle password changing. Pam_cracklib.so will check your new password against rules to ensure that you are selecting a strong password. This module will return

success only if the user selects a valid strong password. The second entry will cause the /etc/passwd and/or /etc/shadow files to be updated once the password has been accepted. The use_authok argument causes the module to use the new password returned from the previous module.

The last line is used to make the appropriate log entries. They record when a user logs in and out. When the pam_pwdb.so is called, it writes the accounting information to syslog and updates both the /etc/utmp and /etc/wtmp files. Pam_pwdb also deals with setting up the user's environment. This might include mounting or unmounting directories, restricting services, or logging. Assuming all modules in the /etc/pam.d/rlogin succeeded, the user has successfully performed a rlogin to the system.

One last structure file worth mentioning is the /etc/security/limits.conf file. The /etc/security directory contains a number of control files that describe PAM's environment and how it operates. The limits.conf file contains limits that PAM should put on individual users or whole groups, excluding root, when they authentication. Setting individual limits has priority over group limits. For example, if limits are set for "test" group, but one of the members of this group has an individual entry, then the user will have his or her limits set according to that entry and not the group limit. In this file, you can limit core sizes, maximum number of logins, address space, and other resources that a user will have available. These limits last only for the single login session.

Summary

The only noted problem with PAM is the additional overhead. On a very busy compute server with thousands of users, the extra resource utilization could cause problems. However, for most systems using PAM will not pose a significant resource issue.

PAM is a definite improvement from *the good old days*. It provides many advantages for a system administrator. PAM enables a common authentication mechanism that can be used with any number of applications. PAM aware applications that are supplied with the operating system distribution, remove the need to have to recompile. The administrator is provided a greater flexibility over the authentication process and related services.

References

Osborn/McGraw-Hill. Red Hat Certified Engineer Linux Study Guide Syngress Media, Inc. 2000

Anonymous. Maximum Linux Security Sams Publishing (2000)

Sun Software White Papers. Sun Solaris Security
http://www.sun.com/software/white-papers/wp-security/#H2_100006

Hernberg, Peter. User Authentication HOWTO (May 2000)
http://mirrors.linuxmall.com/LDP/HOWTO/User-Authentication-HOWTO/x101.html

Morgan, Andrew G. Linux-PAM Systems Administrators Guide http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html

Seifried, Kurt.. PAM – Pluggable Authentication Modules, SysAdmin Magazine (September 2000)

Mann, Scott. Meet PAM Authenticating Users on a Open Source System, Enterprise Linux Magazine http://enterpriselinuxmag.com/public_html/jul00/mann.html (July 2000)

Linux-PAM distribution site http://www.kernel.org/pub/linux/libs/pam/

Sun Microsystems, http://www.sun.com/software/solaris/pam/

Last Updated: August 22nd, 2017

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Tampa - Clearwater 2017	Clearwater, FLUS	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Network Security 2017	Las Vegas, NVUS	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS Dublin 2017	Dublin, IE	Sep 11, 2017 - Sep 16, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MDUS	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, DK	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, GB	Sep 25, 2017 - Sep 30, 2017	Live Event
Data Breach Summit & Training	Chicago, ILUS	Sep 25, 2017 - Oct 02, 2017	Live Event
Rocky Mountain Fall 2017	Denver, COUS	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	The Hague, NL	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Oslo Autumn 2017	Oslo, NO	Oct 02, 2017 - Oct 07, 2017	Live Event
SANS DFIR Prague 2017	Prague, CZ	Oct 02, 2017 - Oct 08, 2017	Live Event
SANS Phoenix-Mesa 2017	Mesa, AZUS	Oct 09, 2017 - Oct 14, 2017	Live Event
SANS October Singapore 2017	Singapore, SG	Oct 09, 2017 - Oct 28, 2017	Live Event
SANS AUD507 (GSNA) @ Canberra 2017	Canberra, AU	Oct 09, 2017 - Oct 14, 2017	Live Event
Secure DevOps Summit & Training	Denver, COUS	Oct 10, 2017 - Oct 17, 2017	Live Event
SANS Tysons Corner Fall 2017	McLean, VAUS	Oct 14, 2017 - Oct 21, 2017	Live Event
SANS Tokyo Autumn 2017	Tokyo, JP	Oct 16, 2017 - Oct 28, 2017	Live Event
SANS Brussels Autumn 2017	Brussels, BE	Oct 16, 2017 - Oct 21, 2017	Live Event
SANS Berlin 2017	Berlin, DE	Oct 23, 2017 - Oct 28, 2017	Live Event
SANS San Diego 2017	San Diego, CAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Seattle 2017	Seattle, WAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Gulf Region 2017	Dubai, AE	Nov 04, 2017 - Nov 16, 2017	Live Event
SANS Miami 2017	Miami, FLUS	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Amsterdam 2017	Amsterdam, NL	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Milan November 2017	Milan, IT	Nov 06, 2017 - Nov 11, 2017	Live Event
Pen Test Hackfest Summit & Training 2017	Bethesda, MDUS	Nov 13, 2017 - Nov 20, 2017	Live Event
SANS Paris November 2017	Paris, FR	Nov 13, 2017 - Nov 18, 2017	Live Event
SANS Sydney 2017	Sydney, AU	Nov 13, 2017 - Nov 25, 2017	Live Event
SANS San Francisco Fall 2017	OnlineCAUS	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced